DEC Project ECE-2023

 bFloat16 Adder/Subtractor

**Dr. Srinivas Boppu**

|  |  |
| --- | --- |
| **NAME** | **Roll no** |
| Taddi Prasanth Sai | 23EC01046 |
| A V Sai Santhosh | 23EC01001 |

**Contents**

[1 Introduction 3](#_Toc196568136)

[2 Problem Statement 3](#_Toc196568137)

[3 BFloat16 Format 3](#_Toc196568138)

[4 Project Design Classification 3](#_Toc196568139)

[5 Detailed Module Description 3](#_Toc196568140)

[5.1 Input Selection 3](#_Toc196568141)

[5.2 Decomposition 3](#_Toc196568142)

[5.3 Exponent Alignment 3](#_Toc196568143)

[5.4 Arithmetic Operation 4](#_Toc196568144)

[5.5 Normalization and Rounding 4](#_Toc196568145)

[5.6 Special Case Management 4](#_Toc196568146)

[5.7 Result Packing 4](#_Toc196568147)

[6 Block Diagram of the Design 4](#_Toc196568148)

[7 FPGA Implementation on Basys 3 Board 4](#_Toc196568149)

[7.1 Development Environment 4](#_Toc196568150)

[7.2 Steps Followed 5](#_Toc196568151)

[7.3 Constraint File Example 5](#_Toc196568152)

[7.4 Testing 5](#_Toc196568153)

[8 Simulation Waveform 5](#_Toc196568154)

[9 Results 6](#_Toc196568155)

[10 Future Work 6](#_Toc196568156)

[11 References 6](#_Toc196568157)

# Introduction

Floating-point arithmetic is essential in AI, scientific computing, and embedded systems. BFloat16 (Brain Floating Point 16-bit) provides high dynamic range and computational efficiency, balancing re- source usage and accuracy.

This project implements a BFloat16 adder/subtractor in Verilog HDL and deploys it on a Basys 3 FPGA board.

# Problem Statement

The aim is to create a hardware module capable of adding and subtracting two BFloat16 numbers. The system must handle normalization, rounding, overflow, underflow, zero, and infinity cases accurately.

# BFloat16 Format

BFloat16 format:

* **Sign:** 1 bit
* **Exponent:** 8 bits (bias 127)
* **Fraction:** 7 bits

Sign (1 bit) *|* Exponent (8 bits) *|* Fraction (7 bits)

Compared to IEEE half-precision (16-bit) format, BFloat16 sacrifices mantissa bits while retaining the same exponent range as FP32, making it ideal for hardware acceleration and AI computations.

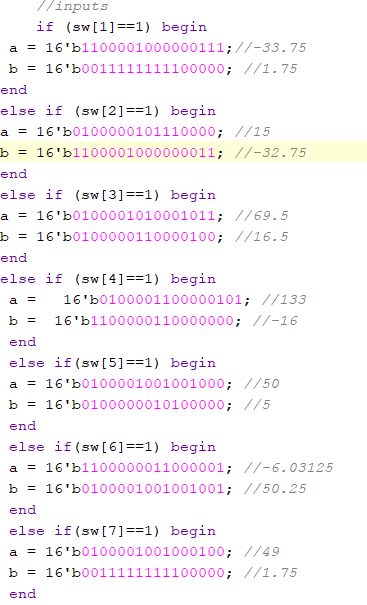
# Project Design Classification

The major functional blocks of the system:

* Operand selection based on user switches
* Operand decomposition (Sign, Exponent, Fraction)
* Exponent alignment
* Conditional addition/subtraction
* Normalization and rounding
* Special cases (Zero, Infinity) handling
* Final packing into 16-bit result

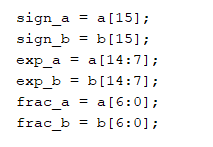
# Detailed Module Description

## Input Selection

Predefined operand pairs are selected based on switch inputs.  
  
  


## Decomposition

Operands are split into sign, exponent, and fraction fields.

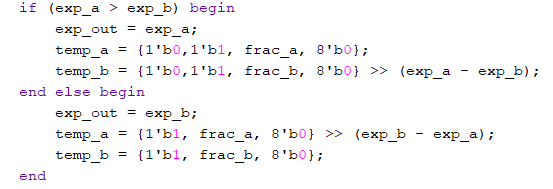


## Exponent Alignment

Fractions are aligned by shifting to match exponents.  
1’b0 is for detecting overflow.  
1’b1 is leading implicit 1   
8’b0 for GRS bits  
>> indicates right shift   
**explanation:**

here we decides the final exponent based on bigger exponent of given two inputs

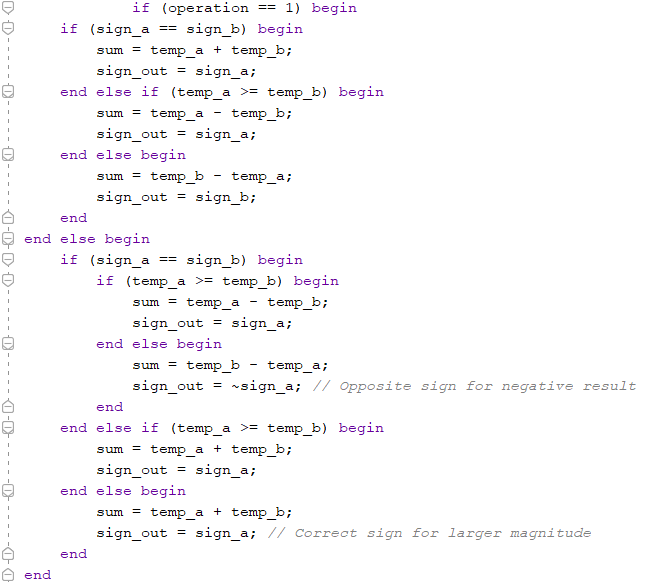
later we have to shift the fraction part of smaller exponent by the exponential difference .



## Arithmetic Operation

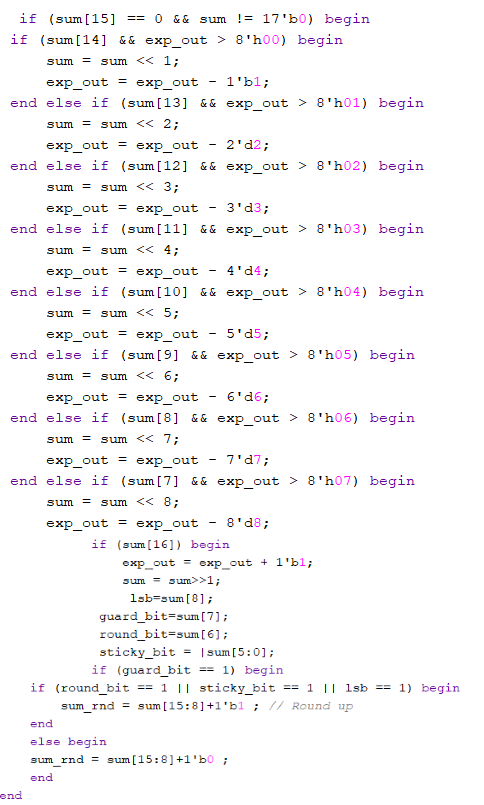
Depending on the operation (add/subtract) and signs, appropriate computation is performed.  
**explanation:**

If operation equals to 1 . it performs addition or else subtraction.

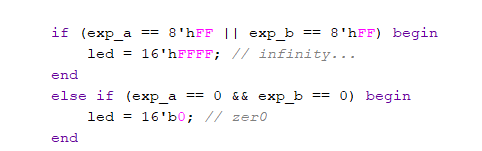


## Normalization and Rounding

Normalization ensures correct format by adjusting leading ones. Rounding uses Guard, Round, Sticky bits to minimize error.  
**explanation:**

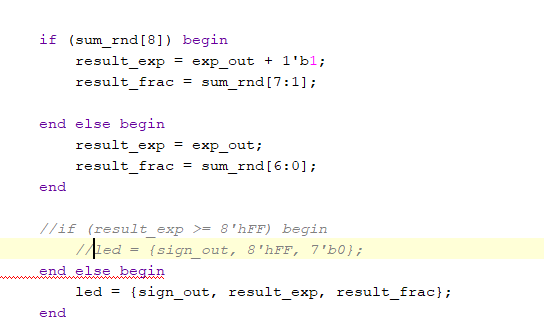
If at all sum[15] ==o then there is a problem that there is no leading implicit 1. So we have to find first 1 in sum and shift it to 15th bit then subtract the amount of shift in final exponent .  
if 16th bit of sum becomes 1 then add 1 to final exponent and right shift the sum by 1 unit.  
for rounding I followed GRS and tie to even rule.  


## Special Case Management

Cases like Zero and Infinity are handled separately.  


## Result Packing

The final output packs sign, exponent, and mantissa back into a 16-bit BFloat16 format, shown via LEDs.



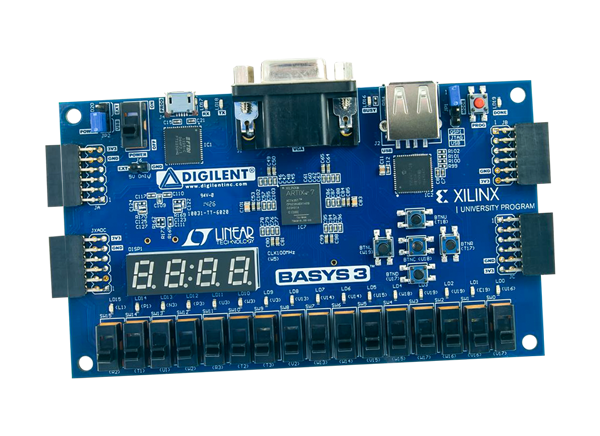
# Block Diagram of the Design

|  |  |
| --- | --- |
| Input Selection | |
|  |  |
| Operand Decomposition | |
|  |  |
| Exponent Alignment | |
|  |  |
| Addition/ Subtraction | |
|  |  |
| Normalization and Rounding | |
|  |  |
| Special Case Handling | |
|  |  |
| Result Packing | |

# FPGA Implementation on Basys 3 Board

## Development Environment

* + - Vivado Design Suite
    - Verilog HDL
    - Basys 3 (Artix-7 FPGA)



## Steps Followed

1. Create Vivado project and select Basys 3.
2. Add Verilog module (bfloatadder.v).
3. Map switches, buttons, LEDs through XDC file.
4. Synthesize, implement, and program FPGA.

## Constraint File

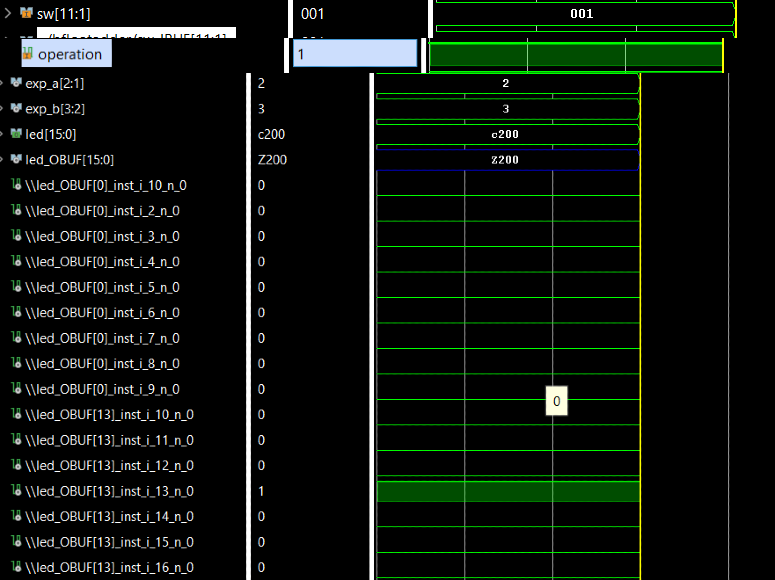


## Testing

* + - Operands selected using switches.
    - Operation controlled via button input.
    - Results observed on LED array.

# Simulation Waveform

Below is a sample placeholder for simulation waveform generated in Vivado simulator.



# Results

The BFloat16 Adder/Subtractor worked successfully on the FPGA, showing correct results for all tested input cases.

# Future Work

* Extend to full IEEE-754 single precision (32-bit floating point).
* Integrate with UART/USB interfaces.
* Introduce pipelining to increase throughput.
* Support for exceptions like NaN (Not-a-Number).

# References

* Jean-Pierre Deschamps, Gery Jean Antoine Bioul, Gustavo D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC, and Embedded Systems*, Wiley, 2006.
* Ashish Reddy Bommana et al., *Design of Synthesis-time Vectorized Arithmetic Hardware for Ta- pered Floating-point Addition and Subtraction*, ACM Transactions on Design Automation of Elec- tronic Systems, 2023. <https://doi.org/10.1145/3567423>
* Vivado Design Suite User Guide, Xilinx Documentation.
* Basys 3 FPGA Board Reference Manual, Digilent Inc.
* Wikipedia Contributors, *BFloat16 Floating-Point Format*, [https://en.wikipedia.org/wiki/Bfloat16\_](https://en.wikipedia.org/wiki/Bfloat16_floating-point_format) [floating-point\_format](https://en.wikipedia.org/wiki/Bfloat16_floating-point_format)
* Stefan Mach. 2021. Floating-point Architectures for Energy-efficient Transprecision Computing. Ph.D. Dissertation. ETH Zurich.
* Stefan Mach. 2021. FPnew- New Floating-Point Unit with Transprecision Capabilities. Retrieved from https://github. com/pulp-platform/fpnew.
* Stefan Mach, Fabian Schuiki, Florian Zaruba, and Luca Benini. 2020. FPnew: An open-source multiformat floating point unit architecture for energy-proportional transprecision computing. IEEE Trans. Very Large Scale Integ. Syst. 29, 4 (2020), 774–787.